VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Scaffold pro Nette Framework

Scaffold for Nette Framework

2014 Martin Drozdek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Drozdek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Scaffold pro Nette Framework
Scaffold for Nette Framework

Zásady pro vypracování:

Cílem této bakalářské práce je vytvořit scaffold systém pro Nette Framework. Před implementací je vhodné prozkoumat existující scaffold systémy u jiných MVC frameworků, například Symphony, Yii, nebo CakePHP a zjistit rozsah jejich funkcionality a UX.
Scaffold systém by měl být schopen generovat entity, presentery a šablony pro CRUD operace. Je také vhodné implemenovat generování nového modulu, presenteru, šablony, formuláře, případně dalších prvků dle podobných systémů. Výsledný program by měl být snadno rozšířitelný a konfigurovatelný.

Seznam doporučené odborné literatury:

GUTMANS, Andi, Stig Sæther BAKKEN a Derick RETHANS. *Mistrovství v PHP 5*. Vyd. 1. Překlad Bogdan Kiszka. Brno: CP Books, 2005, 655 s. ISBN 80-251-0799-X.
Nette Framework: dokumentace [online]. 2013. Dostupné z: http://doc.nette.org/cs/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Daniel Robenek**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 07. května 2014 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 07. května 2014 .....

Tímto bych rád poděkoval vedoucímu práce Ing. Danielu Robenkovi za trpělivost a pomoc při řešení práce.

Abstrakt

Cílem této práce je navrhnout scaffold systém pro Nette framework. V první části se zabývá srovnáním existujících scaffold systémů u jiných MVC frameworků a seznámením s technologiemi použitými při realizaci, ve druhé části pak návrhem a implementací samotné aplikace.

Klíčová slova: Scaffold, Nette Framework, PHP

Abstract

The aim of this work is to propose a scaffold system for the Nette framework. The first part of the work compares existing scaffold systems with other MVC frameworks and introduces technologies used in the implementation. The second part describes the design and implementation of the application itself.

Keywords: Scaffold, Nette Framework, PHP

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
BSD	–	Berkeley Software Distribution
CRUD	–	Create, Read, Update, Delete
DAO	–	Data Access Object
DB	–	Database
GPL	–	General Public License
GUI	–	Graphical User Interface
HTML	–	Hyper Text Markup Language
MIT	–	Massachusetts Institute of Technology
MVC	–	Model-View-Controller
MVP	–	Model-View-Presenter
PC	–	Personal Computer
PHP	–	PHP: Hypertext Preprocessor
SGML	–	Standard Generalized Markup Language
SQL	–	Structured Query Language
WSDL	–	Web Services Description Language
XHTML	–	Extensible HyperText Markup Languagee
XSS	–	Cross-site scripting

Obsah

1 Úvod

Hlavním cílem této bakalářské práce je vytvořit scaffold pro Nette Framework. Scaffold je hodně zjednodušeně řečeno generátor kódu, který šetří čas. Na začátku jsem neměl s Nette Framework ani se scaffoldingem žádné zkušenosti, proto pro mě bylo vytvoření pokročilého nástroje pro tento framework výzvou.

Tvorbě webových aplikací v jazyce PHP se aktivně věnuji již 3 roky, proto od této bakalářské práce očekávám pochopení Nette framework a hlavně pochopení a osvojení si techniky scaffoldingu. Tyto poznatky pak aktivně využiji při tvorbě dalších webových projektů. Aplikaci, kterou při této bakalářské práci vytvořím, poskytnu komunitě, která Nette Framework používá. Doufám, že se u komunity uchytí a stane se oficiálním doplňkem Nette Framework.

V následujících kapitolách budou rozvedeny použité technologie, základní terminologie a ostatní řešení u jiných PHP MVC Frameworků. Ve druhé části této bakalářské práce bude rozebrán návrh, implementace, instalace a funkcionalita vytvořeného scaffold systému.

2 Použité technologie

V této úvodní kapitole prezentuji technologie, které jsem aktivně využil při tvorbě scaffold systému. Kapitola je rozdělena na softwarovou část a část věnující se architektuře.

2.1 Software

2.1.1 PHP

PHP je skriptovací programovací jazyk, který je určen primárně pro programování dynamických webových prezentací a aplikací ve formátu HTML nebo XHTML, nicméně může být použit i pro tvorbu konzolových nebo desktopových aplikací. Zkratka PHP původně znamenala Personal Home Page, ale nyní znamená PHP Hypertext Preprocessor.[1]

Jazyk pracuje na straně serveru, kde je použit PHP Interpret. K uživateli se přenáší výsledek jeho činnosti. Syntaxe jazyka se inspirovala v jazycích jako je C, Perl či Java. Pro tvorbu desktopových či konzolových aplikací se používá kompilovaný forma jazyka. Jazyk podporuje mnoho knihoven a protokolů pro různé účely.

PHP se stal nejrozšířenějším skriptovacím jazykem pro web, hlavně kvůli své jednoduché syntaxi, bohaté zásobě funkcí a hlavně proto, že je šířen pod PHP licencí. PHP licence říká, že software může být nasazen na většině webových serverů a také jako samostatný shell zcela zdarma. Do ledna 2013 byl instalován na více než 240 milionů internetových stránek a 2,1 milionu webových serverů. V PHP bylo vyvinuto mnoho významných webových projektů (např. Wikipedie, WordPress, Drupal, Joomla).

PHP je rozšiřováno celou řadou frameworků, které usnadňují vývoj a umožňují vývojáři soustředit pozornost pouze na své zadání. Mezi nejznámější PHP frameworky patří Zend Framework, CakePHP, Symfony Nette a další.

Výhody:

- Je multiplatformní
- Specializace na webové projekty
- Podpora na hostingových službách
- Jednoduchý na naučení

Nevýhody:

- Je definován pouze svou jednou implementací
- Nekonzistentní pojmenování funkcí
- Chybí debugovací nástroj

2.1.2 HTML

HTML (HyperText Markup Language) je značkovací jazyk pro tvorbu webových stránek. Vznikl z rozsáhlého univerzálního jazyka SGML (Standard Generalized Markup Language). [2]

HTML je základním stavebním kamenem všech webových stránek. Syntaxe je velmi jednoduchá, skládá se z tagů. Tagy jsou nejčastěji párové (například `<div></div>`), ale existují i nepárové (například `<br/>`). V prohlížeči se pak tagy nezobrazují, nýbrž popisují strukturu webové stránky.

Jelikož HTML je součástí každé webové aplikace, přišel jsem s ním do styku i při vývoji scaffoldu. V Nette Framework je podporována verze HTML 5, která sice není oficiálně vydaná, ale hlavní změny všechny prohlížeče v aktuální verzi plně podporují.

2.1.3 Latte

Latte je jazyk pro transformaci textu. Je velmi rychlý a výkonný pro začlenění značky v textových dokumentech. Jazyk definuje velmi jednoduchou syntaxi, která je vhodná pro každý úkol vyžadující textové značky. [3]

S Latte jsem se při tvorbě bakalářské práce seznámil při pronikání do Nette Framework, kdy jsem zjistil, že nepoužívá klasický šablonovací systém PHP jazyka, ale podporuje právě Latte. Syntaxe Latte je opravdu jednoduchá a i složité konstrukce se dají napsat snadně a přehledně. Latte je univerzální jazyk, který nemusí být použit přímo na webu, ale lze jej použít i pro publikační činnost. Lze si vytvářet vlastní makra a ta pak používat napříč dokumenty. V tomto vidím velkou podobnost se systémem Latex, který však míří jen na textové publikace. Důležitým faktorem je skutečnost, že Latte šablony překládá do nativního PHP a ukládá do cache na disk. Díky tomu mají stejný výkon jako PHP šablony, ale v přehlednosti, bezpečnosti a efektivitě jsou o několik řádů dále. Šablony se vždy automaticky vygenerují, jakmile dojde k úpravě zdrojového kódu. V Nette Framework mají Latte šablony rovněž podporu nástroje laděnka, takže při chybě v šabloně víme přesně, na kterém řádku se stala. Latte je také velmi bezpečný, automaticky totiž ošetřuje XSS (Cross Site Scripting) použitím správného escapovaní vypisovaných dat.

2.1.4 MySQL

MySQL je druhý největší široce používaný open-source relační multiplatformní databázový systém. Komunikace probíhá pomocí jazyka SQL. Jedná se o dialekt tohoto jazyka, který má drobné rozšíření. [4]

MySQL je velmi využíván zejména pro webové aplikace. Je optimalizován zejména na rychlost, proto nemá příliš dokonalé způsoby zálohování a donedávna nepodporoval pohledy, triggery a procedury. Velkou výhodu spatřuji v téměř „povinné výbavě“ všech českých poskytovatelů hostingu. Pro vývoj ale nemusíte mít žádný hosting, protože MySQL lze stáhnout a nainstalovat na PC. Mezi desktopové správce MySQL patří například aplikace HeidiSQL. Pro správu databáze je možné použít webového správce phpMyAdmin nebo Adminer. Jelikož Nette Framework podporuje Adminer, využíval jsem právě jeho.

Při tvorbě scaffoldu jsem tento databázový systém upřednostnil, protože je druhým nejpoužívanějším systémem na světě a prvním mezi open source databázovými systémy. Proto si myslím, že bude i v mé aplikaci patřit k nejvíce využívaným. [5]

2.2 Architektura

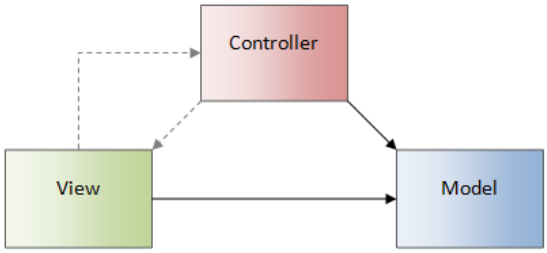
2.2.1 MVC

Model-view-controller je softwarový vzor pro implementaci uživatelských rozhraní. Rozděluje aplikaci do tří vzájemně propojených komponent, které jsou ale na sobě vzájemně nezávislé. Jinými slovy, modifikace jedné komponenty má minimální vliv na ostatní komponenty. MVC byl klíčový pro vývoj GUI (graphics user interface). Původně byl vyvíjen pro stolní počítače, nicméně se velmi rychle přesunul na vývoj webových aplikací, kdy je odpovědnost MVC rozdělena mezi klientem a serverem. [6]

Aplikace, která chce využít architekturu MVC, musí mít implementovány 3 komponenty – Model, View a Controller. Rozdělení komponent lze vidět na Obrázku č.1: MVC.

Model se skládá z dat aplikace a business logiky aplikace. Většinou je realizován více objekty.

View zobrazuje uživatelské rozhraní. Má přímý odkaz na Model, aby mohl zobrazit jeho data. View může, ale nemusí mít vazbu na Controller.



Obrázek 1: MVC [6]

Controller má na starost tok událostí v aplikaci a obecně aplikační logiku. Má přímý odkaz na Model, aby mohl upravit jeho data. Podobně jako u View, i Controller může, ale nemusí mít odkaz na View.

2.2.2 MVP

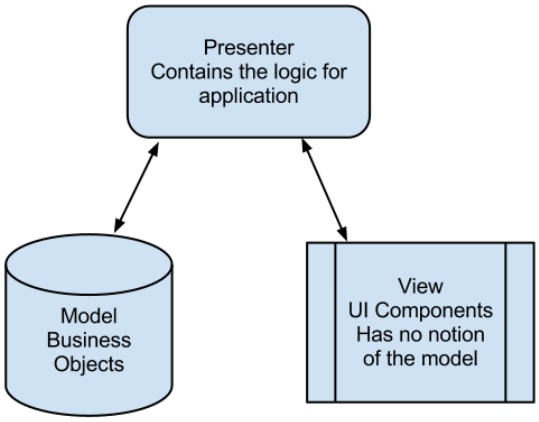
Model-view-presenter je derivát Model-view-controlleru (MVC) používaný především pro tvorbu uživatelského rozhraní. Porovnaní MVP a MVC architektury je znázorněno na Obrázku č. 3: Rozdíl mezi MVC a MVP.

Aplikace, která chce využít architekturu MVP, musí mít implementovány 3 komponenty – Model, View a Presenter. Rozdělení komponent lze vidět na Obrázku č.2: MVP. [7]

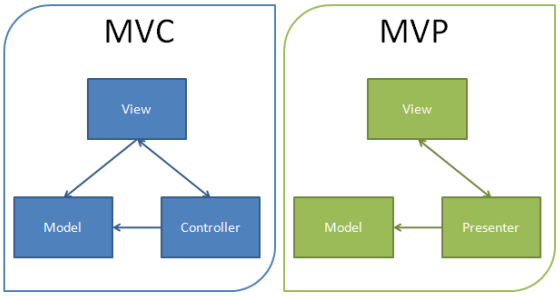
Model je rozhraní definující data aplikace, která mají být zobrazena uživatelskému rozhraní. Model na rozdíl od MVC je striktně doménový. Obecně lze říci, že model je v MVP pouze úložištěm všech dat aplikace bez jakýkoliv funkcí nebo logiky.

View je pasivní rozhraní, které zobrazuje informace získané od presenteru. Také posílá požadavky uživatele na presenter, aby na základě těchto událostí jednal.

Presenter působí jak na modelu, tak na view. Obsahuje veškerou logiku aplikace. Načítá data z modelu, formátuje je, pracuje s nimi a následně je posílá na view. Nebo naopak přijme událost z view, reaguje na ni, načte data z modelu, opět s nimi pracuje a výstup pošle zpět na view.



Obrázek 2: MVP [7]



Obrázek 3: Rozdíl mezi MVC a MVP

3 Scaffold

Scaffolding je technika použitá u některých MVC frameworků, ve kterých programátor napíše specifikaci popisující, jak lze v aplikaci přistoupit ke spojení s databází. Kompilátor poté použije tuto specifikaci k vygenerování kódu, který umožní vytváření, čtení, aktualizaci a odstranění záznamu v databázi (tzv. CRUD).

Scaffolding je nástroj, který usnadňuje práci, šetří čas, umožňuje se soustředit na tvorbu logiky a nadstavby nad vygenerovaným rozhraním a také zamezuje vzniku chyb, protože vývojář nemusí psát tolik kódu.

Tento nástroj se vyvinul z databázových generátorů z dřívějších vývojových prostředí typu klient – server. Při použití scaffoldingu se vygeneruje rozhraní mezi aplikací a databází, pomocí kterého je možné přistupovat k databázi a provádět CRUD operace. [8]

Popularizován byl ve frameworku Ruby on Rails a od té doby byl naimplementován do mnoha jiných frameworků (např. Django, ASP.NET MVC, Symfony, a mnoho dalších). Některé budu popisovat v další části věnované frameworkům.

Většina implementací scaffoldingu je velmi často konfigurovatelná a dále rozšiřitelná, takže lze upravit vygenerovaný kód pro potřeby dané aplikace. Případně lze v některých případech napsat svůj vlastní generátor. Jednotlivé implementace scaffoldingu ve vybraných frameworcích popisuji v kapitole Framework.

4 Framework

Framework je softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji. [9]

Obecně slouží framework k ulehčení práce. Vývojáři se mohou soustředit pouze na logiku aplikace a ne na podpůrný kód, který je však pro běh aplikací nezbytný. Tento podpůrný kód vývojářům poskytne právě framework, protože spoustu logiky již obsahuje a tvůrci aplikací tak nemusí tento kód tvořit od základu znovu.

Dle mého názoru a zkušeností s frameworky si myslím, že použití frameworku znamená méně chyb v aplikaci. Vývojář jednoduše nemusí psát tolik kódu a navíc kód, který poskytuje framework, je prověřený a odladěný. Na druhou stranu by dle mého názoru měli používat framework jen ti programátoři, kteří daný programovací jazyk, nad kterým je framework postaven, již ovládají. Při neznalosti daného programovacího jazyka se může framework stát spíše přítěží, protože nezkušený programátor neví, v čem dělá chybu.

Jsou také odlišné pohledy na frameworky z pozice výkonu aplikací a časové náročnosti. První pohled říká, že frameworky jsou zbytečná zátěž a aplikace s použitím frameworku jednoduše není tak rychlá a efektivní, jako aplikace psaná bez použití frameworku. Jisté také je, že v případě změny frameworku kód napsaný s jeho využitím již nebude funkční. Z hlediska času je nasazení frameworku ze začátku náročnější, protože se programátor s frameworkem musí naučit pracovat. Na druhou stranu ale při použití na velkém projektu nebo na několika projektech se tato nevýhoda změní ve výhodu, protože psaní kódu bude rychlejší, resp. programátor se může soustředit především na daný problém.

Výhody:

- Přímé psaní logiky aplikace
- Úspora času při použití na více projektech

Nevýhody:

- Při přechodu na jiný framework je napsaný kód nepoužitelný
- Aplikace není tak rychlá a efektivní

4.1 Nette Framework

Nette Framework je open source Framework pro tvorbu webových aplikací v jazyce PHP 5. Autorem tohoto frameworku je David Grundl, nyní se již o správu a rozvoj stará organizace Nette Foundation. Nette Framework je nabízený pod licencí GNU GPL. Aktuální verze je verze 2.1.2. [10]

Tento framework má mnoho předností. Zaměřuje se na bezpečnost aplikací a používá technologie, které eliminují výskyt bezpečnostních děr a jejich zneužití, jako je například cross site scripting, session hijacking apod. Velmi užitečným nástrojem Nette Framework je laděnka. Jde o nástroj na odchytávání chyb, který nám jasně říká, na jakém řádku se chyba objevila a poskytuje kompletní informaci o aktuálním stavu aplikace. Výhodou pro české vývojáře je fakt, že Nette Framework má nejaktivnější komunitu v České republice, takže je možno se na oficiálním fóru zeptat na jakékoliv nejasnosti. Také existuje množství doplňků, které je velmi snadné použít v aplikacích a disponuje rozsáhlou dokumentací psanou v českém jazyce.

Nette Framework má bohaté zastoupení moderními technologiemi na webu (např. dependency injection, kiss, mvc, dry, ajax). Také má velmi promyšlený objektový návrh, díky čemuž může využívat nových vlastností verze PHP 5 i samostatných komponent a tím pádem vede vývojáře k dobrému návrhu aplikací s důrazem na budoucí rozšiřitelnost.

4.1.1 Scaffold v Nette Framework

Scaffold v tomto frameworku zcela chybí. Proto doufám, že po dokončení mé bakalářské práce se objeví jako možný doplněk v oficiální knihovně doplňků Nette Framework.

4.2 Symfony Framework

Symfony je webový aplikační Framework pro MVC aplikace psané v PHP 5. Tento framework je open source a je vydán pod licencí MIT. Původní název tohoto frameworku byl Sensio Framework, odvozený od firmy Sensio Labs, která Framework vyvíjela a neustále sponzoruje jeho další vývoj. Aktuální verze tohoto frameworku je 2.4.4. [11]

Výhodou tohoto frameworku jsou velmi nízké nároky na výkon, protože se používá bytecode cache. Vývojáře vede k čistému návrhu aplikací a dává jim kompletní kontrolu nad konfigurací.

Symfony na rozdíl od Nette Frameworku používá pro konfigurační soubory jazyk YAML, pro šablony šablonovací jazyk Twig. Podobně jako Nette Framework má i Symfony

dobrou podporu ladicích nástrojů, velké množství již existujících komponent a rozsáhlou komunitu.

Zvláštností frameworku Symfony je, že k vytváření projektů, aplikací modulů a dalších nastavení je nutné používat příkazový řádek. Tento fakt může způsobovat problém při použití tohoto frameworku u některých poskytovatelů hostingu.

4.2.1 Scaffold v Symfony

Modul pro scaffold se v Symfony jmenuje SensioGeneratorBundle. Tento generátor je schopný vygenerovat kostru kódu, formuláře a nebo CRUD controllery na základě schématu Doctrine 2.

4.2.1.1 Funkcionalita

První funkcí je vytváření holé kostry kódu. V tomto režimu se vygeneruje nová struktura a automaticky se aktivuje v rámci aplikace. Druhou možností je vygenerování doctrine entity, definice mapování a třídní proměnné včetně getterů a setterů. Třetí možností je vygenerování CRUD controllerů pro doctrine entitu, které nám umožní provádět základní operace na modelu. Jedná se o výpis všech záznamů, zobrazení detailu záznamu, vytvoření nového záznamu, úpravu a smazání existujícího záznamu. Poslední možností je generování třídy formuláře mapující určitou doctrine entitu.

4.2.1.2 UX

Veškerá obsluha scaffoldu v Symfony probíhá z příkazového řádku, přičemž příkazy se zadávají interaktivně, jinými slovy scaffold klade otázky a uživatel doplňuje textový vstup. Tento interaktivní režim lze vypnout a napsat rovnou výsledný příkaz.

4.3 CakePHP Framework

CakePHP je open source webový aplikační framework napsaný v jazyku PHP, je založen na MVC a je distribuován pod licencí MIT. Tento framework vyvinul Michal Tatarynowicz, nyní se však o jeho správu a rozvoj stará nadace Cake Software. Aktuální verze je verze 2.4.7. [12]

Podobně jako Nette Framework i CakePHP klade důraz na zabezpečení aplikací, nabízí tedy vestavěné nástroje pro ochranu (např. proti SQL injection nebo cross site scripting). Vývojáře nabádá k dodržování čisté MVC konvence. Dále jim šetří práci, protože obsahuje generátor kódu a scaffold systém pro vytváření rychlých základů aplikací. CakePHP

nabízí již v základu velké množství funkcí, jako jsou například překlady, cachování, validace, autentifikace atd.

Na rozdíl od Nette Framework nebo Symfony v CakePHP nejsou konfigurační soubory, pouze se nastaví přístup k databázi a framework je schopný fungovat.

4.3.1 Scaffold v CakePHP

Scaffold systém v CakePHP je odlišný od Symfony. Pro jeho použití je nutné mít již vytvořený model a controller. Controller je prázdný, musí dědit z AppController a pouze se do něj zapíše příkaz, kde se má vygenerovat výsledný kód. Na základě jména controlleru se použije daný model.

4.3.1.1 Funkcionalita

Jedinou možností je generování formulářů pro editaci a přidání a také funckionality pro výpis všech entit, mazání a výpis detailu entity. Nabízí také možnost vygenerování šablon se směrováním, to znamená, že budou vytvořeny ve stejném adresáři, nicméně přístupné budou z jiné adresy v prohlížeči. Dále je zde možnost si upravit šablony, které se používají pro generování formuláře, výpisu všech záznamů a výpisu detailu záznamu.

4.3.1.2 UX

Na rozdíl od Symfony je v CakePHP místo příkazového řádku použito psaní příkazů přímo do kódu aplikace. Pro přidání scaffoldu do aplikace stačí v daném controlleru vložit proměnnou `$scaffold`. Po vygenerování kódu je potřeba nastavit vazby v modelech, jinak například ve formulářích budou vidět pouze id navázaných modelů.

4.4 Yii Framework

Yii je webový aplikační MVC framework psaný v jazyku PHP, který je striktně objektově orientovaný. Tento Framework je open source a je šířen pod licencí BSD. Na vývoji se podílí mezinárodní tým vývojářů. Aktuální verze je verze 1.1.4. [13]

Předností frameworku Yii je jeho striktně MVC přístup, který rozděluje aplikaci do jednotlivých vrstev a je proto vhodný pro velké komplexní webové aplikace. Dále nabízí podporu DAO (Database Object Access) a Active Record, Query Builder a DB Migration, což umožnuje vývojářům modelovat databázi na úrovni objektů. Yii podporuje také webové služby a nabízí automatické generování WSDL. Také nabízí generátory kódů včetně scaffoldu, kterému se níže budu věnovat podrobněji. Yii nabízí vrstvenou cache,

která výrazně snižuje reakční čas aplikace. Podporuje také skiny a šablony, pomocí kterých lze velmi snadno přepínat vzhled aplikace. V neposlední řadě Yii implementuje knihovnu jQuery a nabízí velké množství ajaxových komponent.

Podobně jako Nette Framework se také Yii zaměřuje na bezpečnost, proto nabízí technologie proti cross site scriptingu, SQL injection apod. Pro Yii existuje rozsáhlý seznam komponent, které jednoduše rozšiřují výsledné aplikace. Validace formulářů a obecně práce s formuláři je rovněž podobná jako u Nette Framework.

4.4.1 Scaffold v Yii

Pro scaffolding je Yii vybaven nástrojem Gii, který vychází z předchozí yiic shell, ale nyní již není podporován. Yiic shell se ovládal z příkazového řádku, kdežto Gii je webově založen. To znamená, že vše se děje na webu. Ke Gii se přistoupí přes url adresu, ve formuláři se navolí přesný typ výsledného kódu a vyplní se nezbytné informace pro generaci. Dle mého názoru má Yii uživatelsky nejpřívětivější scaffold ze všech frameworků zkoumaných v této bakalářské práci.

4.4.1.1 Funkcionalita

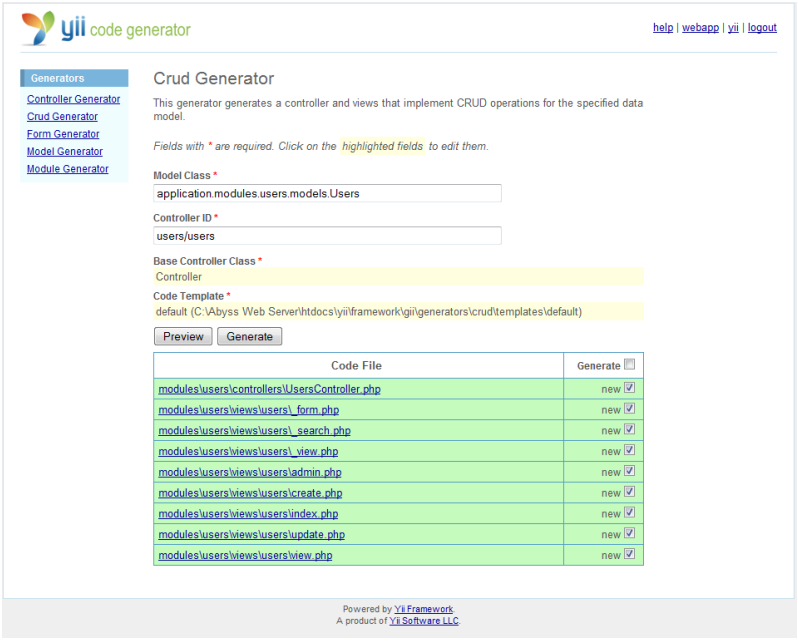
V Gii je několik generátorů kódů, které jsou zodpovědné pouze za svou práci. Tyto generátory jsou Controller generator, CRUD generator, Form generator, Model generator a Module generator. Gii lze dále rozšiřovat a upravovat šablony, podle kterých se kód generuje. Rozšiřování je možné napsáním vlastního generátoru kódu. Takto vytvořený generátor musí splňovat přesné požadavky dle typu generátoru, aby bezchybně spolupracoval s Gii.

4.4.1.2 UX

Pro použití Gii je nutné komponent nejdříve zaregistrovat v nastavení. Důležité je, že scaffolding ve výchozím stavu funguje pouze na localhostu. Pro webový přístup je nutné nakonfigurovat příslušné IP adresy. V základním nastavení je postup uživatele následující: zadá url pro přístup ke Gii; vyplní parametry na webu s příslušnými parametry; poté má možnost zobrazit náhled, jak bude vygenerovaný kód vypadat. Je-li vše v pořádku, klikne na tlačítko generate a kód se vygeneruje.

4.5 Zend Framework 2

Zend Framework 2 je objektově orientovaný webový aplikační Framework implementovaný v jazyce PHP 5. Podobně jako veškeré popsané frameworky v této



Obrázek 4: Náhled Gii v Yii

bakalářské práci je i tento open source a je šířen pod licencí BSD. Zend Framework 2 je evolucí Zend Framework 1, který měl přes 15 milionů stažení. Hlavním sponzorem projektu je firma Zend Technologies. Aktuální verze je verze 2.3.1. [14]

Komponenty v tomto frameworku jsou plně objektově orientované a s několika málo závislostmi na jiných komponentách. Všechny společně užívají objektový princip SOLID. Podobně jako u jiných frameworků tedy vývojář může do své aplikace zahrnout množství již vytvořených komponentů. Velká podpora je rovněž ve webových službách, kde framework nabízí komponent ZendService. S tímto komponentem se vývojáři jednoduše mohou připojit k webovým službám. Jako jeden z mála PHP frameworků je Zend Framework podporován ze stranyGoogle a Microsoft, kteří se podílejí na poskytování webových služeb a dalších technologií usnadňujících vývojářům práci.

Podobně jako u Nette Frameworku je i zde podpora formulářů, komponentů na autentifikace, autentizaci atd. Rovněž je zde rozsáhlá komunita vývojářů a podrobná dokumentace. Zend Framework nabízí vývojářům možnost získat oficiální certifikaci Zend Framework 2 Certified Architect.

4.5.1 Scaffold v Zend Framework 2

Scaffold v Zend Frameworku 2 je realizován jako jedna z možností rozsáhlého nástroje ZFTool. Tento systém však nelze považovat za plnohodnotný scaffold systém, protože nepodporuje CRUD generaci controllerů, modelů ani formulářů. Jediná generace, kterou ZFTool provede, spočívá ve vytvoření nového projektu, případně modulu.

4.5.1.1 Funkcionalita

Tento modul umí vytvořit nový projekt a základní kostru kódu, nový modul v již existující aplikaci a nainstalovat knihovny a správu konfiguračních souborů aplikací v Zend Frameworku 2.

4.5.1.2 UX

Veškeré ovládání scaffoldu probíhá pomocí příkazového řádku. Podobně je ovládání řešeno ve frameworku Symfony.

5 Specifikace požadavků

V této kapitole stručně shrnuji klíčové požadavky na výsledný scaffold systém pro Nette Framework.

5.1 Generování entit

Aplikace musí umět generovat entity. Jako základní databázový systém bude použito MySQL. Musí umět zpracovávat základní databázové typy VARCHAR, INT, FLOAT, TEXT, BOOL, DATE a DATETIME. Dále bude vygenerován model pro příslušnou entitu, který bude mít na starost práci s databází.

5.2 Generování modulu

Scaffold systém bude umět vygenerovat samostatný modul, do kterého se poté vygenerují presentery a šablony. Také musí zvládnout generovat nové moduly do již existujících modulů.

5.3 Generování presenteru

Presentery budou vygenerovány do příslušného modulu a budou rozděleny dle akcí na presenter pro výpis všech entit, editaci entity, přidání entity a výpis detailu entity.

5.4 Generování šablon

Podobně jako u presenteru budou i šablony vygenerovány do příslušného modulu a budou rovněž rozděleny na šablony pro výpis všech entit, editaci entity, přidání entity a výpis detailu entity.

5.5 Generování formulářů

Pro editaci a přidání entity budu vygenerovány formuláře, které dle Best Practices frameworku Nette Framework budou jako komponenty, generovat se tedy bude továrnička na tyto komponenty. [15]

5.6 Konfigurace

Aplikace musí být velmi snadno konfigurovatelná. Je nutno konfigurovat podobu modelu, presenterů a šablon. Formuláře a pohledy musí být konfigurovatelné až na úroveň typu databázového sloupce.

5.7 Ostatní

Výsledná aplikace bude snadno rozšiřitelná. Modely, presentery, šablony a formuláře budou moci být vygenerovány samostatně na základě informací v databázi.

6 Návrh

V této kapitole se budu věnovat návrhu scaffold systému pro Nette Framework.

6.1 Ovládání aplikace

Ovládání navrženého scaffold systému bude probíhat přes příkazový řádek. Uživatel bude muset zadat cestu ke složce, ve které se scaffold nachází, a za ní uvést příslušné parametry. Více se této tématice věnuji v kapitole Funkcionalita.

Za ovládání a chod celého scaffold systému je zodpovědná třída `Scaffold`, kterou popisuji v kapitole Implementace.

6.2 Generátor modulu

Generátor modulu slouží k vytvoření samostatného funkčního celku, který Nette Framework nazývá moduly. Moduly se obecně skládají z kompletní uzavřené logiky, obsahují tedy presentery, šablony a modely. Pro potřebu mé aplikace jsem do generování nového modulu zakomponoval pouze presentery, šablony a formulář. Modul může být vygenerován do již existujícího modulu.

Za generaci modulu je zodpovědná třída `ModuleGenerator`, kterou popisuji v kapitole Implementace.

6.3 Generátor modelu

Generátor modelů slouží k vytvoření modelu generované entity. Model se bude generovat dle připravené šablony. Tato šablona je editovatelná, takže si každý vývojář může výsledný model upravit dle svého uvážení.

V základní šabloně model dědí z Nette\Object. Obsahuje konstruktor, ve kterém dochází k předání reference na databázi. Dále obsahuje veřejné metody `find`, `save` a `delete`. Metoda `find` způsobuje vyhledání dané entity. Metoda `save` umožňuje entitu uložit, v případě že daná entita neexistuje tak ji vytvoří. Metoda `delete` slouží k vymazání entity. Šablona je zobrazena v příloze Šablony v části Šablony pro model ve Výpisu 1: Šablona Model. Výsledný vygenerovaný model je zobrazen v příloze Vygenerovaný kód v části Vygenerovaný model ve Výpisu 12: Vygenerovaný model.

Za generaci modelu je zodpovědná třída `ModelGenerator`, kterou popisuji v kapitole Implementace.

6.4 Generátor presenterů

Generátor presenterů slouží k vytvoření presenterů generované entity. Tyto presentery se generují dle připravených editovatelných šablon. Vývojář si tyto šablony může upravit dle svého uvážení.

V mém návrhu aplikace je presenter pro danou entitu rozdělen na více presenterů dle logiských částí, kterou obstarávají. Presenter je tedy rozdělen na `BasePresenter`, `BaseEntityPresenter`, `AddPresenter`, `EditPresenter`, `ListPresenter` a `DetailPresenter`.

`BasePresenter` dědí z `BasePresenter` předchozího modulu, pokud tento existuje, jinak z `BasePresenter` aplikace. Šablona, podle které se tento presenter generuje, je zobrazena v příloze Šablony v části Šablony pro presentery ve Výpisu 2: Šablona BasePresenter. Vygenerovaný BasePresenter lze vidět v příloze Vygenerovaný kód v části Vygenerované presentery ve Výpisu 13: Vygenerovaný BasePresenter.

`BaseEntityPresenter` dědí z `BasePresenteru` a obsahuje konstruktor pro přístup k databázi. Entity v názvu se vždy nahradí jménem generované entity. Například pro entitu `auto` bude tento presenter `BaseAutoPresenter`. Šablona, podle které se tento presenter generuje, je zobrazena v příloze Šablony v části Šablony pro presentery ve Výpisu 3: Šablona BaseEntityPresenter. Vygenerovaný BaseEntityPresenter lze vidět v příloze Vygenerovaný kód v části Vygenerované presentery ve Výpisu 14: Vygenerovaný BaseEntityPresenter.

`AddPresenter` obsahuje logiku pro vytvoření formuláře pro vkládání, uložení a také smazání entity. Dědí z `BaseEntityPresenteru`. Šablona, podle které se tento presenter generuje, je zobrazena v příloze Šablony v části Šablony pro presentery ve Výpisu 4: Šablona AddPresenter. Vygenerovaný AddPresenter lze vidět v příloze Vygenerovaný kód v části Vygenerované presentery ve Výpisu 15: Vygenerovaný AddPresenter.

`EditPresenter` obsahuje logiku pro vytvoření formuláře pro editaci a následně její uložení. Dědí z `BaseEntityPresenteru`. Šablona, podle které se tento presenter generuje, je zobrazena v příloze Šablony v části Šablony pro presentery ve Výpisu 5: Šablona EditPresenter. Vygenerovaný EditPresenter lze vidět v příloze Vygenerovaný kód v části Vygenerované presentery ve Výpisu 16: Vygenerovaný EditPresenter.

`ListPresenter` obsahuje logiku pro výpis všech entit entit a také konstruktor, který obsahuje logiku pro přístup k databázi. Dědí z `BaseEntityPresenteru`. Šablona, podle které se tento presenter generuje, je zobrazena v příloze Šablony v části Šablony pro presentery ve Výpisu 6: Šablona ListPresenter. Vygenerovaný ListPresenter lze vidět v

příloze Vygenerovaný kód v části Vygenerované presentery ve Výpisu 17: Vygenerovaný ListPresenter.

`DetailPresenter` obsahuje logiku pro detailní výpis dané entity. Dědí z `BaseEntityPresenteru`. Šablona, podle které se tento presenter generuje, je zobrazena v příloze Šablony v části Šablony pro presentery ve Výpisu 7: Šablona DetailPresenter. Vygenerovaný DetailPresenter lze vidět v příloze Vygenerovaný kód v části Vygenerované presentery ve Výpisu 18: Vygenerovaný DetailPresenter.

Za generaci presenterů je zodpovědná třída `PresenterGenerator`, kterou popisuji v kapitole Implementace.

6.5 Generátor šablon

Generátor šablon slouží k vytvoření šablon generované entity. Šablony jsou vygenerovány na základě připravených editovatelných šablon. Vývojář si tyto šablony může upravit dle svého uvážení.

Šablony v Nette Framework složí jako view v MVC, resp. MVP architektuře. V mém návrhu počítám s generováním několika šablon dle logických celků. Výsledné šablony tedy jsou šablony pro přidání entity a editaci entity, výpisu všech entit a výpisu detailního pohledu entity.

V šabloně pro přidání a editaci prvku je pouze odkaz na formulář, který je rozebrán níže v části Generátor formuláře. Šablony, podle kterých se tyto šablony generují, jsou zobrazeny v příloze Šablony v části Šablony pro šablony ve Výpisu 8: Šablona šablony pro přidání a ve Výpisu 9: Šablona šablony pro editaci. Vygenerované šablony lze vidět v příloze Vygenerovaný kód v části Vygenerované šablony ve Výpisu 19: Vygenerovaná šablona pro přidání a ve Výpisu 20: Vygenerovaná šablona pro editaci. Screenshot šablony pro přidání je zobrazen na Obrázku 7: Vygenerovaný formulář.

V šabloně pro detail je výpis jako tabulka, ve které jsou všechny atributy dané entity. Dále je zde navigační prvek pro návrat na výpis všech entit. Šablona, podle které se tato šablona generuje, je zobrazena v příloze Šablony v části Šablony pro šablony ve Výpisu 10: Šablona šablony pro detailní výpis. Vygenerovanou šablonu lze vidět v příloze Vygenerovaný kód v části Vygenerované šablony ve Výpisu 21: Vygenerovaná šablona pro detailní výpis. Screenshot šablony pro detailní výpis je zobrazen na Obrázku 5: Vygenerovaná šablona pro detail.

V šabloně pro výpis všech entit je výpis jako tabulka, ve které jsou všechny entity. Dále jsou zde navigační prvky na přidání entity a editaci dané entity, výpisu detailu dané

Obrázek 5: Vygenerovaná šablona pro detail

entity a smazaní dané entity. Šablona, podle které se tato šablona generuje, je zobrazena v příloze Šablony v části Šablony pro šablony ve Výpisu 11: Šablona šablony pro výpis všech entit. Vygenerovanou šablonu lze vidět v příloze v části Vygenerované šablony ve Výpisu 22: Vygenerovaná šablona pro výpis všech entit. Screenshot šablony pro výpis všech entit je zobrazen na Obrázku 6: Vygenerovaná šablona pro výpis.

Výpis v šabloně detailu a všech entit nemusí být tabulka, ale může být použitý jakýkoliv jiný grid pro výpis. Stačí upravit šablony, dle kterých se šablona pro detail nebo výpis všech entit generují.

Za generaci šablon je zodpovědná třída `TemplateGenerator`, kterou popisuji v kapitole Implementace.

6.6 Generátor formuláře

Generátor formuláře slouží k vytvoření formuláře, který se používá pro editaci a přidání generované entity. Formuláře jsou vygenerovány na základě připravených editovatelných šablon. Vývojář si tyto šablony může upravit dle svého uvážení.

Formuláře jsou vytvářeny dle Best Practise Nette Frameworku [15]. Místo vygenerování dvou samostatných formulářů pro editaci a přidání entity se vygeneruje továrnička na formuláře. Tato továrnička se následně volá v `AddPresenteru` v případě

Obrázek 6: Vygenerovaná šablona pro výpis

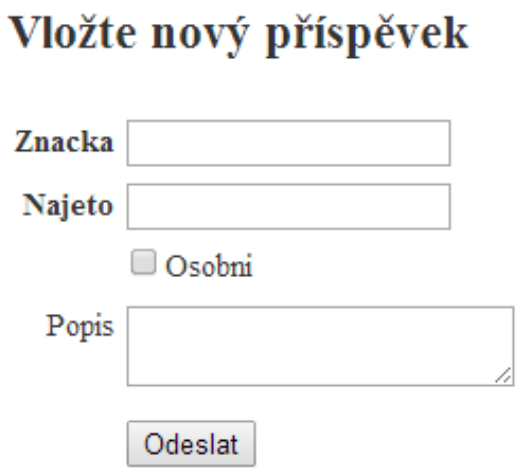
přidání entity a v `EditPresenteru` v případě editace entity. Továrnička je vygenerována dle šablon, které jsou odlišné dle typu atributu entity. Například pro typ varchar se vygeneruje klasický input, ale pro typ bool se vygeneruje checkbox. Při generaci těchto jednotlivých polí do továrničky je také důležité, zda může být daný atribut Null a jaká je jeho velikost. Pokud daný atribut nemůže být null, nastaví se ve formuláři, že tento prvek musí být vyplněn. V případě, že má atribut nastavenou velikost, nastaví se také do pole ve formuláři jako nejvyšší možná hodnota. Dále se v poli hlídá, zda je zadaná hodnota stejného typu jako daný atribut. Například pro typ int nelze do pole ve formuláři zadat string.

Vygenerovaná továrnička na formulář je zobrazena v příloze Vygenerovaný kód v části Vygenerovaná továrnička na formulář ve Výpisu 23: Vygenerovaná továrnička na formulář.

Za generaci šablon je zodpovědná třída `FormGenerator` a jednotlivé třídy pro typ sloupce `VarcharGenerator`, `IntGenerator`, `FloatGenerator`, `TextGenerator`, `DateGenerator`, `DatetimeGenerator` a `BoolGenerator`, které popisuji v kapitole Implementace.

6.7 Generátor entity

Generátor entity slouží k vytvoření generované entity v databázi aplikace. Nejprve dojde k získání spojení s databází a následně k vytvoření entity, resp. tabulky v databázi. Jednotlivé atributy entity mohou v mém návrhu nabývat hodnot int, float, varchar, text, bool, date a datetime. Pokud toto uživatel nedodrží, nebude entita vytvořena. Jednotlivým



Obrázek 7: Vygenerovaný formulář

atributům se také dle uživatelského vstupu nastaví defaultní hodnota, maximální velikost, a zda může být null či nikoliv.

Za generaci šablon je zodpovědná třída `MySQLDatabase` a jednotlivé třídy pro typ sloupce `VarcharGenerator`, `IntGenerator`, `FloatGenerator`, `TextGenerator`, `DateGenerator`, `DatetimeGenerator` a `BoolGenerator`, které popisuji v kapitole Implementace.

7 Implementace

V následující kapitole budu popisovat implementaci navrženého scaffold systému. Zabývat se budu především nejdůležitějšími částmi aplikace, které jsou pro navržený scaffold zásadní. Tyto části podrobně popíšu a vysvětlím.

7.1 Třída Scaffold

Třída `Scaffold` je hlavní třídou aplikace. Je zodpovědná za získání uživatelského vstupu, jeho rozparserování na dané typy a zavolání příslušných metod. Třída má dvanáct privátních metod a jeden konstruktor.

V konstruktoru dochází k přečtení parametrů z uživatelského vstupu a zavolání jedné z metod `help`, `generate` nebo `regenerate`.

Metoda `help` vypisuje nápovědu pro scaffold systém pro Nette Framework.

Metoda `generate` způsobí kompletní vygenerování modulu, presenterů, modelu, šablon, továrničky na formuláře a dané entity do databáze. Metoda nejprve zavolá metodu na rozparserování parametrů, které se načetly v konstruktoru. Poté postupně volá příslušné metody ke generování již zmíněných částí.

Metoda `regenerate` způsobí pouze částečnou generaci dle uživatelského vstupu. Nejprve dojde k rozparserování parametrů, které se načetly v konstruktoru. Poté se třída připojí do databáze a vyhledá danou entitu. Z této entity načte příslušné parametry pro jednotlivé atributy entity. Tyto parametry se použijí pro regeneraci částí, které uživatel zadal. Dále metoda zavolá příslušné metody pro generaci daných částí.

V metodách `generateModule`, `generateModel`, `generatePresenters`, `generateTemplates` a `generateForm` dochází k volání příslušných tříd pro generaci jednotlivých částí kódu.

Metody `readArgv`, `writeConsole` a `processCode` jsou pomocné metody, které se využívají při volání metod `generate`, resp. `regenerate`.

Poslední metoda je metoda `loadParamsMysql`. Tato metoda způsobí načtení parametrů atributů pro příslušnou entitu. Tato metoda se volá v metodě `regenerate`.

7.2 Třída Generator

Třída `Generator` je bázová třída všech tříd aplikace, které mají ve jménu Generator, s výjimkou třídy `ModuleGenerator`. Třída obsahuje jedenáct chráněných metod, které

využívají jednotlivé třídy pro generaci daných komponent.

Metoda `getRealPath` vrací umístění, kde se nachází scaffold aplikace. Tohoto se využívá při načítání šablon pro jednotlivé komponenty, které se nacházejí ve složce templates.

Metoda `loadTemplate` s parametrem `$path` načítá šablonu z umístění, které se přenáší v parametru path. V této metodě dochází k volání metody `getRealPath` a dochází k načtení šablony, která se vrací jako string.

Metoda `write` s parametry `$template` a `$file` zapisuje vygenerovaný kód, který se přenáší v parametru `$template`, do souboru, který dostane v parametru `$file`.

Metoda `replaceTemplateString` s parametry `$whereReplace`, `$whatReplace` a `$withReplace` způsobí nahrazení textu, který se přenáší v parametru `$whatReplace`, textem, který se přenáší v parametru `$withReplace`, v celkovém textu, který je přenášen v posledním parametru `$whereReplace`. Po nahrazení metoda vrací výsledný string. Tato metoda se používá k nahrazování proměnných v šabloně za vygenerované proměnné pro příslušnou generovanou entitu.

Metody `getModulePath`, `getModuleNamespace`, `getModules` a `getCurrentModule` slouží k práci s moduly. Pomocí těchto metod dojde k nalezení cesty, získání namespace, získání cesty pro Nette Framework a získání aktuálního modulu. Tohoto se využívá pro generování jednotlivých presenterů, šablon a formuláře.

Metody `getSize`, `getNull` a `getDefault` slouží k nalezení velikosti atributu, k zjištění, zda může být atribut null, a k nastavení výchozí hodnoty pro atribut generované entity. Tohoto využívají třídy `VarcharGenerator`, `IntGenerator`, `FloatGenerator`, `TextGenerator`, `BoolGenerator`, `DateGenerator` a `DatetimeGenerator` k získání příslušných vlastností pro daný atribut generované entity.

7.3 Třída ModuleGenerator

Třída `ModuleGenerator` je zodpovědná za vytvoření adresářů v modulu. Třída obsahuje jednu veřejnou metodu `create` a jednu privátní metodu `createDir`.

Privátní metoda `createDir` je zodpovědná za vytvoření daného adresáře. Nejprve zkontroluje, zda daný adresář existuje. Pokud adresář neexistuje, tak ho vytvoří. Dále metoda `createDir` vrací bool hodnotu, a to podle toho, zda se adresář podařilo vytvořit či nikoliv. V případě, že adresář již existuje, vrací TRUE.

Veřejná metoda `create` zajistí vytvoření celého adresáře, čímž je modul nachystán pro generování dalších prvků. Metoda `create` má dva parametry: modul, který má být vytvořen, a název generované entity. Metoda vrací bool hodnotu, podle toho, zda akce proběhla korektně nebo selhala.

7.4 Třída ModelGenerator

Třída `ModelGenerator` je zodpovědná za vytvoření modelu generované entity. Třída dědí z třídy `Generator`, která je popsána výše v této kapitole. Třída `ModelGenerator` má jednu veřejnou metodu `create` a druhou privátní metodu `loadEntityAtributes`.

Privátní metoda `loadEntityAtributes` je zodpovědná ze převedení názvů atributů entity do textové podoby. Má jeden parametr, ve kterém je předáno dvourozměrné pole přenášející atributy entity. Metoda vrací textový řetězec obsahující jména atributů oddělené čárkou.

Veřejná metoda `create` zajistí vytvoření modelu. Metoda má dva parametry: název generované entity a dvourozměrné pole se všemi atributy generované entity. Metoda nejdříve provede nahrání šablony modelu, poté dojde k nahrazení proměnné `[scaffold-entityName]` za jméno generované entity. Následně se zavolá privátní metoda `loadEntityAtributes` a dojde k nahrazení proměnné `[scaffold-entityAtributesComma]`. Nakonec se výsledný kód uloží. Metoda vrací bool podle toho, zda došlo k uspěšnému nebo neúspěšnému uložení modelu. V metodě jsou použity metody zděděné, které popisuji výše u třídy `Generator`.

7.5 Třída PresenterGenerator

Třída `PresenterGenerator` je zodpovědná za vytvoření presenterů pro generovanou entitu. Třída dědí z třídy `Generator`, kterou popisuji výše v této kapitole. Třída `PresenterGenerator` má jednu veřejnou metodu `create` a šest privátních metod `createBasePresenter`, `createBaseEntityPresenter`, `createAddPresenter`, `createEditPresenter`, `createListPresenter` a `createDetailPresenter`.

Veřejná metoda `create` zajišťuje vytvoření presenteru daného typu. Metoda má tři parametry: modul, ve kterém se entita generuje, jméno entity a jméno metody. V metodě se nejdříve inicializují proměnné nutné k zavolání privátní metody, a to podle daného parametru. Poté se zavolá jedna z již zmíněných metod. Tyto metody vrací bool, který vrací i metoda `create`.

Všechny privátní metody mají podobnou implementaci, proto zde popíšu pouze jednu z nich, například `createAddPresenter`. Metoda nejdříve načte šablonu AddPresenteru. Poté dojde k nahrazení proměnných v šabloně za skutečné údaje. Na závěr metoda uloží vygenerovaný AddPresenter a vrátí bool podle toho, zda došlo k úspěšnému nebo neúspěšnému uložení. V metodě jsou použity metody zděděné, které popisuji výše u třídy `Generator`.

7.6 Třída TemplateGenerator

Třída `TemplateGenerator` je zodpovědná za vytvoření šablon pro generovanou entitu. Třída dědí z třídy `Generator`, kterou popisuji výše v této kapitole. Třída `TemplateGenerator` má jednu veřejnou metodu `create` a sedm privátních metod `createAddTemplate`, `createEditTemplate`, `createListTemplate`, `createDetailTemplate`, `loadEntityList`, `loadHeaderTable` a `loadEntityDetail`.

Veřejná metoda `create` zajišťuje vytvoření šablony daného typu. Metoda má čtyři parametry: modul, ve kterém se entita generuje, jméno entity, pole atributů entity a jméno metody. V metodě se nejdříve inicializují proměnné nutné k zavolání privátní metody, a to podle daného parametru. Poté se zavolá jedna z metod pro vytvoření šablony. Tyto metody jsou `createAddTemplate`, `createEditTemplate`, `createListTemplate` a `createDetailTemplate`, které vrací bool a ten je následně vrácen i metodou `create`.

Zmíněné metody pro vytvoření šablony mají podobnou implementaci, proto zde popíšu pouze jednu z nich, například `createAddTemplate`. Metoda nejdříve načte šablonu Add. Poté dojde k nahrazení proměnných v šabloně za skutečné údaje. Na závěr metoda uloží vygenerovanou šablonu a vrátí bool, a to podle toho, zda došlo k úspěšnému nebo neúspěšnému uložení.

Metody `loadEntityList`, `loadHeaderTable` a `loadEntityDetail` jsou pomocné metody k vytvoření šablon pro výpis všech entit a výpis detailu entity.

Všechny privátní metody používají metody zděděné, které popisuji výše u třídy `Generator`.

7.7 Třída FormGenerator

Třída `FormGenerator` je zodpovědná za vytvoření továrničky na tvorbu formulářů pro generovanou entitu. Princip tohoto faktu je vysvětlen v kapitole Návrh, části Generátor

formuláře. Tato třída dědí z třídy `Generator`, kterou popisuji výše v této kapitole. Třída má jednu veřejnou metodu `create` a jednu privátní metodu `loadEntityField`.

Veřejná metoda `create` zajišťuje vytvoření továrničky formulářů pro přidání a editaci generované entity. Metoda má tři parametry: modul, ve kterém se entita generuje, jméno entity a pole atributů entity. V této metodě dochází nejprve k načtení informací o modulech, které jsou klíčové pro volání dalších metod z této metody. Následně se volá privátní metoda `loadEntityFields`. Poté dochází k načtení šablony a nahrazení šablonových proměnných vygenerovaným kódem. Na závěr metoda uloží vygenerovanou továrničku na formuláře a vrátí bool, a to podle toho, zda došlo k úspěšnému nebo neúspěšnému uložení.

Privátní metoda `loadEntityFields` je zodpovědná za zavolání a načtení formulářového pole pro atributy, které se předávají v parametru atributes. Tato metoda vrací kompletní kód formulářových polí jako string.

V obou metodách se používají metody zděděné, které popisuji výše u třídy `Generator`.

7.8 Ostatní třídy

Ostatní třídy implementované v rámci navrženého scaffold systému jsou třídy `DBConnector`, `MySQLDatabase`, `IntGenerator`, `VarcharGenerator`, `TextGenerator`, `FloatGenerator`, `BoolGenerator`, `DateGenerator` a `DatetimeGenerator`.

Třída `DBConnector` je zodpovědná za získání parametrů potřebných k vytvoření spojení s databází. Tyto parametry načítá z konfiguračních souborů v Nette Frameworku.

Třída `MySQLDatabase` je zodpovědná za vytvoření databázové tabulky a vytvoření databázových sloupců dle uživatelského vstupu. Používá k tomu třídy, které popisuji níže.

Třídy `IntGenerator`, `VarcharGenerator`, `TextGenerator`, `FloatGenerator`, `BoolGenerator`, `DateGenerator` a `DatetimeGenerator` mají všechny čtyři metody. První metoda je `generateMySQL`, která vrací databázový sloupec dle typů třídy. Této metody se využívá pří vytváření tabulky v databázi ve tříde `MySQLDatabase`. Druhá metoda je `generateForm`. Tato metoda vrací formulářové pole pro danou třídu a volá se ze třídy `FormGenerator`. Třetí metoda je metoda `generateGridList`. Tato metoda vrací textovou podobu dle typu pro výpis všech entity, používá ji třída `TemplateGenerator`. Poslední metodou je `generateGridDetail`, která vrací

opět textovou podobu dle typu pro výpis detailu entity. Rovněž ji využívá třída `TemplateGenerator`.

8 Instalace

V následující kapitole budu popisovat instalaci navrženého scaffold systému. Při dodržení všech zmíněných bodů bude výsledný scaffold systém připravený ke spuštění. Veškeré funkce popisuji v kapitole Funkcionalita.

8.1 Prerekvizity

Pro běh navrženého scaffold systému je nutné mít stažený Nette Framework verze 2.1.2. Dále je nutné mít korektně nainstalované PHP ve verzi 5. Také je nutné mít nastavenou databázi MySQL a mít nastaveny dsn, user a password v lokální konfiguraci projektu, tedy config.local.neon.

8.2 Samotná instalace a spuštění

Jelikož dle mého názoru jsou nejlepší pokyny k instalaci krátké, jasné a výstižné, shrnul jsem je do následujících bodů:

1. Najeďte na adresu `https://github.com/MartinDrozdek/scaffold`.
2. Z této adresy stáhněte zip soubor. Tohoto docílíte kliknutím na tlačítko download zip v pravém dolním rohu.
3. Soubor rozbalte kdekoliv na disku.
4. Z rozbalené složce otevřete složku scaffold-master.
5. Z této složky zkopírujte složku scaffold.
6. Nyní najeďte do kořenové složky, ve které máte nainstalovaný projekt v Nette Framework (složka ve které vidíte složky app, bin, www atd.).
7. Do této složky vložte složku zkopírovanou v bodě 5.
8. V této složce držte shift a klikněte pravým tlačítkem myši.
9. V zobrazeném dialogu klikněte na `Zde otevřít příkazové okno`.
10. Ve spuštěném příkazovém řádku zadejte příkaz `php scaffold/index.php ?`
11. Tímto jste spustili nápovědu pro scaffold systém pro Nette Framework.
12. Všechny příkazy se tedy zadávají za `php scaffold/index.php` a oddělují se mezerou.

8.3 Testování

Celý scaffold systém jsem testoval na systému Windows 7. Celý systém jsem testoval pouze na localhostu. Při dodržení již zmíněných prerekvizit a instalačních pokynů bude navržený scaffold systém pro Nette Framework korektně fungovat.

9 Funkcionalita

V této kapitole shrnu funkcionalitu navrženého scaffold systému pro Nette Framework.

9.1 Možnosti

9.1.1 Základní podporované typy

Základní podporované typy atributů entity jsou:

- varchar
- int
- float
- bool
- text
- date
- datetime

Tyto typy přímo vychází z MySQL. Zápis vždy vypadá tak, že napíšeme jméno atributu a poté typ atributu. Všechna jména atributu musí být bez diakritiky.

Například když chci atribut `jméno` typu varchar bude příkaz vypadat `jmeno:varchar`

9.1.2 Doplňkové informace k typu

Doplňkové informace jsou ty informace, které lze k typu atributu přiřadit. Tyto informace se oddělují od typu rovnítkem (=). Mezi sebou se pak oddělují čárkou (,). Hodnota se odděluje pomlčkou (-). Doplňkové informace nemusí být ve stejném pořadí.

Možné doplňkové informace jsou:

- size – jako hodnota je brána celé číslo.
- default – jako hodnota je brán řetezec
- null – jako hodnota je bráno buď true, pokud typ může být null, nebo false, pokud nemůže být null.

Například budu-li chtít vygenerovat atribut `jméno` typu varchar s velikostí 10, defaultně vyplněným Name a bude moci být null napíšu příkaz: `jmeno:varchar=size-10,default-Name,null-true`

9.1.3 Generace do modulu

Je možné generovat entitu do již existujícího modulu. To lze udělat tak, že za slovo entity je dopsáno „module:“ a cestu k modulu ze složky app. Tedy, chci-li generovat entitu audi do modulu auto, bude příkaz vypadat takto: `php scaffold/index.php entity module:AutoModule audi ...`

9.2 Generování entity

Základním příkazem pro generování entity je příkaz `entity`. Pří této funkci dojde k vygenerování modulu, modelu, presenterů, šablon a továrničky na formuláře a k vytvoření entity v databázi. Důležité je manuálně zaregistrovat továrničku na formuláře do konfigurace Nette Frameworku. Nejlépe je to vidět na následujícím příkladu.

9.2.1 Příklad

Budu chtít vygenerovat entitu auto, která bude mít tyto atributy:

- Značka – typ varchar, velikost bude 50, a nebude moci být null
- Najeto – typ int, velikost bude 10, defaultně bude 10000 a nebude moci být null
- Osobní – typ bool, defaultně bude nastaveno true
- Popis – typ text, může být null

Tuto entitu vygeneruji příkazem: `php scaffold/index.php entity Auto znacka:varchar=size-50,null-false najeto:int=size-10,default-10000,null-false osobni:bool=default:TRUE popis:text=null-true`

Nyní je již kód vygenerovaný a entita vytvořená. Poslední co je třeba udělat je zaregistrovat továrničku na formuláře jako službu do konfigurace Nette Frameworku.To lze provést následujícím postupem:

1. otevřete ve složce app/config soubor config.neon
2. najděte položku services

3. připište jako poslední řádek text, který se zobrazí v příkazovém řádku.

V našem případu je to tedy: `-\App\AutoModule\AutoFormFactory`

9.3 Regenerování požadované části

V případě změně v databázové tabulce dané entity je žádoucí, aby šli jednotlivé části aplikace jednoduše regenerovat. Navržený scaffold systém toto podporuje.

9.3.1 Možnosti

Regenerovat můžeme následující části aplikace:

- Model – příkaz `Model`
- Presentery – příkaz `Presenters`
- Šablony – příkaz `Templates`
- Továrničku na formulář – příkaz `Form`
- Model + továrničku na formulář + šablony – příkaz `Entity`

9.3.2 Příkazy

Základní příkaz pro regeneraci je příkaz `regenerate`. Za ním následuje část aplikace, kterou chceme regenerovat. Dále se očekává modul, ve kterém je entita vygenerovaná. Pokud není v žádném modulu, není třeba nic psát a lze pokračovat jménem entity.

9.3.3 Příklad

V databázi u tabulky auto přidám sloupec majitel. Entita není v žádném modulu. Aby se mi aplikace přizpůsobila změně v databázi, stačí provést jednoduchý příkaz: `php scaffold/index.php regenerate Entity auto`

9.4 Nápověda

Se základní použitelností pomůže zabudovaná nápověda. Vyvolání probíhá příkazem `?`. Celý příkaz bude vypadat `php scaffold/index.php ?`

10 Závěr

V předchozích kapitolách jsem nejprve popsal jednotlivé použité technologie, poté jsem zanalyzoval scaffold systémy v jiných PHP frameworcích, ze kterých jsem vycházel při návrhu aplikace. Následně jsem popsal návrh a implementaci aplikace. V poslední části se věnuji používání této aplikace, tedy instalaci a funkcionalitě.

Pří vývoji jsem narazil na řadu problémů, z nich většinu jsem zdárně vyřešil, nicméně některé jsem vyřešit nestihl. Nejzásadnějším dosud nedokončeným problémem je dynamické zaregistrování továrničky na formuláře, aby uživatel nemusel toto dělat ručně.

V současné době při podmínkách, které jsem shrnul v kapitole instalace, je výsledná aplikace plně funkční. Nicméně je pravděpodobné, že během následujícího používání mohou nastat stavy, se kterými jsem nepočítal.

Do budoucna bych chtěl aplikaci vylepšit především o automatickou registraci továrničky na formuláře, přidat další databázové typy a další typy podporovaných databází.

11 Literatura

[1] *PHP [online]*, poslední úpravy 03.05.2014.
Dostupné z: http://en.wikipedia.org/wiki/Php

[2] *HTML [online]*, poslední úpravy 02.05.2014.
Dostupné z: http://en.wikipedia.org/wiki/Html

[3] *Latte [online]*, poslední úpravy 06.11.1999.
Dostupné z: http://www.latte.org/latte.html

[4] *MySQL [online]*, poslední úpravy 25.04.2014.
Dostupné z: http://en.wikipedia.org/wiki/Mysql

[5] *Srovnání databázvých enginů [online]*, poslední úpravy 01.05.2014.
Dostupné z: http://db-engines.com/en/ranking

[6] *MVC [online]*, poslední úpravy 07.05.2009.
Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[7] *MVP [online]*, poslední úpravy 28.04.2014.
Dostupné z: http://en.wikipedia.org/wiki/Model-view-presenter

[8] *Scaffold [online]*, poslední úpravy 04.05.2014.
Dostupné z: http://en.wikipedia.org/wiki/Scaffold_(programming)

[9] *Framework [online]*, poslední úpravy 11.02.2014.
Dostupné z: http://cs.wikipedia.org/wiki/Framework

[10] *Nette Framework [online]* Dostupné z: http://nette.org/

[11] *Symfony Framework [online]* Dostupné z: http://symfony.com/

[12] *CakePHP Framework [online]* Dostupné z: http://cakephp.org/

[13] *Yii Framework [online]* Dostupné z: http://www.yiiframework.com/

[14] *Zend Framework 2 [online]* Dostupné z: http://framework.zend.com/

[15] *Best practise: Formuláře jako komponenty [online]*
Dostupné z: http://pla.nette.org/cs/best-practise-formulare-jako-komponenty

[16] GUTMANS, Andi, Stig Sæther BAKKEN a Derick RETHANS. *Mistrovství v PHP 5 Vyd. 1. Překlad Bogdan Kiszka.* Brno: CP Books, 2005.

Seznam obrázků

Seznam výpisů zdrojového kódu

A Šablony

A.1 Šablona pro model

```
<?php
namespace App\Model;

use Nette;

class [scaffold-entityName] extends Nette\Object{

    private $db;

    public function __construct(Nette\Database\Context $database) {
    $this->db = $database;
    }

    public function find($id) {
    $result = $this->db->fetch('SELECT [scaffold-entityAtributesComma] FROM
    `[scaffold-entityName]` WHERE `id` = ?', $id);
    if ($result == FALSE) {
        throw new ModelException("[scaffold-entityName] s id $id neexistuje.");
    } else {
        return $result;
    }
    }

    public function save($id,$values) {
    if ($id == NULL) {
        $this->db->query('INSERT INTO `[scaffold-entityName]`', $values);
        return $this->db->getInsertId();
    } else {
        return $this->db->query('UPDATE `[scaffold-entityName]` SET ? WHERE id = ?',
          $values, $id);
    }
    }

    public function delete($id) {
    return $this->db->query('DELETE FROM `[scaffold-entityName]` WHERE `id`=?',$id);
    }
}
?>
```

Výpis 1: Šablona Model

A.2 Šablony pro presentery

```
<?php
namespace App\[scaffold-namespace]\Presenters;

use Nette, App\Model;

class BasePresenter extends \App\[scaffold-previousModule]Presenters\BasePresenter{

}
?>
```

Výpis 2: Šablona BasePresenter

```
<?php
namespace App\[scaffold-namespace]\Presenters;

use Nette, App\Model;

class Base[scaffold-entityName]Presenter extends BasePresenter{

    protected $database;

    /** @persistent */
    public $[scaffold-entityName]Id;

    public $[scaffold-entityName];

    public $[scaffold-entityName]Values;

    public function __construct(Nette\Database\Context $database) {
   $this->database = $database;

    }

    public function startup() {
   parent::startup();
   $this->[scaffold-entityName] = new Model\[scaffold-entityName]($this->database);
   if ($this->[scaffold-entityName]Id != "") {
        try {
       $this->[scaffold-entityName]Values = $this->[scaffold-entityName]->find
       ($this->[scaffold-entityName]Id);
        } catch (ModelException $exc) {
       $this->flashMessage("[scaffold-entityName] s id $this->[scaffold-entityName]Id
      nebyl nalezen.", 'error');
```

```
        $this->redirect(':[scaffold-Modules]:List:');
         }
    }
     }

     public function beforeRender() {
    parent::beforeRender();
    $this->template->item = $this->[scaffold-entityName]Values;
     }


}
?>
```

Výpis 3: Šablona BaseEntityPresenter

```
<?php

namespace App\[scaffold-namespace]\Presenters;

use Nette, App\Model;

class AddPresenter extends Base[scaffold-entityName]Presenter{

    /** @var \App\[scaffold-namespace]\[scaffold-entityName]FormFactory @inject */
    public $[scaffold-entityName]FormFactory;

    public function createComponent[scaffold-entityName]Form(){
    $form = $this->[scaffold-entityName]FormFactory->create();
    $form->addSubmit('Add');
    return $form;
     }

     public function onSubmitAdd($form){
    $values = $form->getValues();
    $id = $this->[scaffold-entityName]->save(NULL,$values);
    $this->flashMessage('[scaffold-entityName] byl pridan.','success');
    $this->redirect(':[scaffold-Modules]:Detail:default',array(' [scaffold-entityName]Id' => $id));
     }

     public function actionAdd(){
    $form = $this->getComponent('[scaffold-entityNameSmall]Form');
    $form->onSubmit[] = callback($this, 'onSubmitAdd');
     }
```

```
    public function actionDelete($id){
    $this->[scaffold-entityName]->delete($id);
    $this->flashMessage('[scaffold-entityName] byl smazan.','success');
    $this->redirect(':[scaffold-Modules]:List:');
    }
}
?>
```

Výpis 4: Šablona AddPresenter

```
<?php

namespace App\[scaffold-namespace]\Presenters;

use Nette, App\Model;

class EditPresenter extends Base[scaffold-entityName]Presenter{

    /** @var \App\[scaffold-namespace]\[scaffold-entityName]FormFactory @inject */
    public $[scaffold-entityName]FormFactory;

    public function createComponent[scaffold-entityName]Form(){
    $form = $this->[scaffold-entityName]FormFactory->create();
    $form->addSubmit('Edit');
    return $form;
    }

    public function onSubmitEdit($form){
    $values = $form->getValues();
    $this->[scaffold-entityName]->save($this->[scaffold-entityName]Id, $values);
    $this->flashMessage('[scaffold-entityName] byl upraven.','success');
    $this->redirect(':[scaffold-Modules]:Detail:default', array(':[scaffold-entityName]Id' =>
    $this->[scaffold-entityName]Id));
    }

    public function actionEdit($[scaffold-entityName]Id){
    $form = $this->getComponent('[scaffold-entityNameSmall]Form');
    $form->setDefaults($this->[scaffold-entityName]Values);
    $form->onSubmit[] = callback($this, 'onSubmitEdit');
    }
}
?>
```

Výpis 5: Šablona EditPresenter

```
<?php

namespace App\[scaffold-namespace]\Presenters;

use Nette, App\Model;

class ListPresenter extends BasePresenter{
        private $database;

        public function __construct(Nette\Database\Context $database) {
        $this->database = $database;
        }
    public function renderDefault(){
            $this->template->items = $this->database->table('[scaffold-entityName]');
    }
}
?>
```

Výpis 6: Šablona ListPresenter

```
<?php

namespace App\[scaffold-namespace]\Presenters;

use Nette, App\Model;

class DetailPresenter extends Base[scaffold-entityName]Presenter{

    public function renderDefault($[scaffold-entityName]Id){

    }
}
?>
```

Výpis 7: Šablona DetailPresenter

A.3 Šablony pro šablony

```
{block content}

<h2>Vlozte novy prispevek</h2>

{control [scaffold-form]}
```

Výpis 8: Šablona šablony pro přidání

```
{block content}

<h2>Upravte prispevek</h2>

{control [scaffold-form]}
```

Výpis 9: Šablona šablony pro editaci

```
{block content}

<p><a n:href=":[scaffold-Modules]:List: [scaffold-entityName]Id => null">zpet na vypis prispevku</a></p>

<h2>Detail zaznamu</h2>

[scaffold-entityDetail]
```

Výpis 10: Šablona šablony pro detailní výpis

```
{block content}

<p><a n:href=":[scaffold-Modules]:Add:Add">Pridat novou polozku</a></p>

<h2>Vypis vsech</h2>

[scaffold-entityList]
```

Výpis 11: Šablona šablony pro výpis všech entit

B Vygerovaný kód

Vygenerovaný kód pro ukázkový příklad v kapitole Funkcionalita v části Generování entity.

B.1 Vygenerovaný model

```
<?php
namespace App\Model;

use Nette;

class Auto extends Nette\Object{

    private $db;

    public function __construct(Nette\Database\Context $database) {
   $this->db = $database;
    }

    public function find($id) {
   $result = $this->db->fetch('SELECT znacka, najeto, osobni, popis FROM 'Auto' WHERE 'id' = ?', $id);
   if ($result == FALSE) {
       throw new ModelException("Auto s id $id neexistuje.");
   } else {
       return $result;
   }
    }

    public function save($id,$values) {
   if ($id == NULL) {
       $this->db->query('INSERT INTO 'Auto'', $values);
       return $this->db->getInsertId();
   } else {
       return $this->db->query('UPDATE 'Auto' SET ? WHERE id = ?', $values, $id);
   }
    }

    public function delete($id) {
   return $this->db->query('DELETE FROM 'Auto' WHERE 'id'=?', $id);
    }
}
?>
```

Výpis 12: Vygenerovaný model

B.2 Vygenerované presentery

```
<?php
namespace App\AutoModule\Presenters;

use Nette, App\Model;

class BasePresenter extends \App\Presenters\BasePresenter{

}
?>
```

Výpis 13: Vygenerovaný BasePresenter

```
<?php
namespace App\AutoModule\Presenters;

use Nette, App\Model;

class BaseAutoPresenter extends BasePresenter{

    protected $database;

    /** @persistent */
    public $AutoId;

    public $Auto;

    public $AutoValues;

    public function __construct(Nette\Database\Context $database) {
    $this->database = $database;
    }

    public function startup() {
    parent::startup();
    $this->Auto = new Model\Auto($this->database);
    if ($this->AutoId != "") {
        try {
        $this->AutoValues = $this->Auto->find($this->AutoId);
```

```
        } catch (ModelException $exc) {
        $this->flashMessage("Auto s id $this->AutoId nebyl nalezen.", 'error');
        $this->redirect(':Auto:List:');
        }
    }
    }

    public function beforeRender() {
    parent::beforeRender();
    $this->template->item = $this->AutoValues;
    }

}
?>
```

Výpis 14: Vygenerovaný BaseEntityPresenter

```
<?php

namespace App\AutoModule\Presenters;

use Nette, App\Model;

class AddPresenter extends BaseAutoPresenter{

    /** @var \App\AutoModule\AutoFormFactory @inject */
    public $AutoFormFactory;

    public function createComponentAutoForm(){
    $form = $this->AutoFormFactory->create();
    $form->addSubmit('Add');
    return $form;
    }

    public function onSubmitAdd($form){
    $values = $form->getValues();
    $id = $this->Auto->save(NULL,$values);
    $this->flashMessage('Auto byl pridan.','success');
    $this->redirect(':Auto:Detail:default',array('AutoId' => $id));
    }

    public function actionAdd(){
    $form = $this->getComponent('autoForm');
    $form->onSubmit[] = callback($this, 'onSubmitAdd');
    }
```

```
    public function actionDelete($id){
    $this->Auto->delete($id);
    $this->flashMessage('Auto byl smazan.','success');
    $this->redirect(':Auto:List:');
    }
}
?>
```

Výpis 15: Vygenerovaný AddPresenter

```
<?php

namespace App\AutoModule\Presenters;

use Nette, App\Model;

class EditPresenter extends BaseAutoPresenter{

    /** @var \App\AutoModule\AutoFormFactory @inject */
    public $AutoFormFactory;

    public function createComponentAutoForm(){
    $form = $this->AutoFormFactory->create();
    $form->addSubmit('Edit');
    return $form;
    }

    public function onSubmitEdit($form){
    $values = $form->getValues();
    $this->Auto->save($this->AutoId, $values);
    $this->flashMessage('Auto byl upraven.','success');
    $this->redirect(':Auto:Detail:default', array('AutoId' => $this->AutoId));
    }

    public function actionEdit($AutoId){
    $form = $this->getComponent('autoForm');
    $form->setDefaults($this->AutoValues);
    $form->onSubmit[] = callback($this, 'onSubmitEdit');
    }
}
?>
```

Výpis 16: Vygenerovaný EditPresenter

```
<?php

namespace App\AutoModule\Presenters;

use Nette, App\Model;

class ListPresenter extends BasePresenter{

  private $database;

  public function __construct(Nette\Database\Context $database) {
      $this->database = $database;
  }

    public function renderDefault(){
      $this->template->items = $this->database->table('Auto');
    }
}
?>
```

Výpis 17: Vygenerovaný ListPresenter

```
<?php

namespace App\AutoModule\Presenters;

use Nette, App\Model;

class DetailPresenter extends BaseAutoPresenter{

    public function renderDefault($AutoId){

    }
}
?>
```

Výpis 18: Vygenerovaný DetailPresenter

B.3 Vygenerované šablony

```
{block content}

<h2>Vlozte novy prispevek</h2>
```

```
{control autoForm}
```

Výpis 19: Vygenerovaná šablona pro přidání

```
{block content}

<h2>Upravte prispevek</h2>

{control autoForm}
```

Výpis 20: Vygenerovaná šablona pro editaci

```
{block content}

<p><a n:href=":Auto:List: AutoId => null">zpet na vypis prispevku</a></p>

<h2>Detail zaznamu</h2>

<table>
    <tr>
    <td>
   Znacka
    </td>
    <td>
   {$item->znacka}
    </td>
 </tr><tr>
    <td>
   Najeto
    </td>
    <td>
   {$item->najeto}
    </td>
 </tr><tr>
    <td>
   Osobni
    </td>
    <td>
   {$item->osobni}
    </td>
 </tr><tr>
    <td>
   Popis
    </td>
```

```
    <td>
   {$item->popis}
    </td>
 </tr>
</table>
```

Výpis 21: Vygenerovaná šablona pro detailní výpis

```
{block content}

<p><a n:href=":Auto:Add:Add">Pridat novou polozku</a></p>

<h2>Vypis vsech</h2>

<table>
    <tr>
   <td>
    Znacka
</td><td>
    Najeto
</td><td>
    Osobni
</td><td>
    Popis
</td>
   <td>
       Actions
   </td>
    </tr>
    {foreach $items as $item}
   <tr>
       <td>
    {$item->znacka}
</td><td>
    {$item->najeto}
</td><td>
    {$item->osobni}
</td><td>
    {$item->popis}
</td>
        <td>
       <a href="{link :Auto:Detail:default $item->id}">View</a>
       <a href="{link :Auto:Edit:edit $item->id}">Edit</a>
       <a href="{link :Auto:Add:delete $item->id}">Delete</a>
        </td>
```

```
    </tr>
    {/foreach}
</table>
```

Výpis 22: Vygenerovaná šablona pro výpis všech entit

B.4 Vygenerovaná továrnička na formulář

```
<?php

namespace App\AutoModule;

use Nette\Application\UI;

class AutoFormFactory extends \Nette\Object{

    public function create(){
        $form = new UI\Form;
        $form->addText('znacka', 'Znacka')->addRule(UI\Form::MAX_LENGTH,'Delka muze byt
            maximalne 50 znaku',50)->setRequired('Toto pole je povinne');$form->addText
            ('najeto', 'Najeto')->addRule(UI\Form::INTEGER,'Zadana hodnota musi byt
            celociselna')->addRule(UI\Form::MAX_LENGTH,'Delka muze byt maximalne 10 cislic',
            10)->setRequired('Toto pole je povinne');$form->addCheckbox('osobni', 'Osobni');
            $form->addTextArea('popis', 'Popis');
        return $form;
    }

}
?>
```

Výpis 23: Vygenerovaná továrnička na formulář

C CD

Na přiloženém CD se nachází zdrojové kódy scaffold systému pro Nette Framework a tato bakalářská práce v elektronické podobě.